UNITED STATES PATENT AND TRADEMARK OFFICE

mw

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/714,198	11/14/2003	Long Li	42P17832	2667

8791 7590 07/23/2007
BLAKELY SOKOLOFF TAYLOR & ZAFMAN
1279 OAKMEAD PARKWAY
SUNNYVALE, CA 94085-4040

EXAMINER	
VU, TUAN A	

ART UNIT	PAPER NUMBER
2193	

MAIL DATE	DELIVERY MODE
07/23/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)
	10/714,198	LI ET AL.
	Examiner	Art Unit
	Tuan A. Vu	2193

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *14 November 2003*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-36* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-36* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *14 November 2003* is/are: a)☐ accepted or b)☒ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *8/24/05*.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application

6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the application filed 11/24/03.

Claims 1-36 have been submitted for examination.

Drawings

2. The drawings are objected to under 37 CFR 1.83(a) because they fail to show:

(i) 'hoist instructions no longer needed' (step 550, Fig. 9);

(ii) 'sink instructions no longer needed' (step 579, Fig. 12) as described in the specification.

The Specifications does not mention about the phrase 'no longer needed' anywhere in the process as to establish what instructions are to sink or hoist; nor does the Specifications convey a concept of need when processing a list of candidate instructions.

Any structural detail that is essential for a proper understanding of the disclosed invention should be shown in the drawing. MPEP § 608.02(d).

Corrected drawing sheets in compliance with 37 CFR 1.121(d) are required in reply to the Office action to avoid abandonment of the application. Any amended replacement drawing sheet should include all of the figures appearing on the immediate prior version of the sheet, even if only one figure is being amended. The figure or figure number of an amended drawing should not be labeled as "amended." If a drawing figure is to be canceled, the appropriate figure must be removed from the replacement sheet, and where necessary, the remaining figures must be renumbered and appropriate changes made to the brief description of the several views of the drawings for consistency. Additional replacement sheets may be necessary to show the renumbering of the remaining figures. Each drawing sheet submitted after the filing date of an application must be labeled in the top margin as either "Replacement Sheet" or "New Sheet" pursuant to 37 CFR 1.121(d). If the changes are not accepted by the examiner, the applicant will be notified and informed of any required corrective action in the next Office action. The objection to the drawings will not be held in abeyance.

Specification

3. The disclosure is objected to because of the following informalities:

- Figure 15C as mentioned in the Specifications, is not per se listed in the 'Brief Description of the Drawings" section.

- The phrase 'an instruction **is that is** hoisted outside of an outmost…' (pg. 16, para 00064) requires typographical correction.
- The phrase 'among the bas**ing** blocks until sinking instructions …' (pg. 15, para 00059) requires typographical correction.

Appropriate correction is required.

Claim Objections

4. Claims 1, 11, 32, 35 are objected to because of the following informalities: 'bonding instructions' -- in association with 'boundary instructions'. The 'bonding instruction' is strongly suggestive of a typographical error or misuse of language, because *bonding* is clearly different from *boundary*. Lack of antecedent basis can be applied as a 35 USC 112 type of rejection in case no correction is made.

5. Claims 32-33, 35-36 are objected to for the following improper syntax: 'wherein the compiler to cause building…' , 'wherein the compiler to cause hoisting' (line 1) and suggested proper grammatical construct for those phraseswould be: 'wherein the compiler is operable to cause'

Appropriate correction is required.

Claim Rejections - 35 USC § 112

6. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

7. Claims 3-9, 13-19, 33 and 36 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. That is, claims 3-9, 13-19, 33 and 36 are rejected under 35

U.S.C. 112, second paragraph, as being incomplete for omitting essential elements, such omission amounting to a gap between the elements. See MPEP § 2172.01. The omitted elements are: relationship between 'within the nodes' limitation with respect to *hoisting* or *sinking*; i.e. the spatial relationship between a given CFG loop block in which candidate instructions are hoisted with respect to the rest of the loop or other part of the CFG; OR the spatial relationship between a block wherein a candidate instruction is sunk with respect to the rest of the CFG.

Specifically, *hoisting* code is a well-known software optimization methodology requiring a directional movement of code leaving one source to another destination; and *sinking* an instruction also requires movement integrating an instruction (from a source) into a new destination. Claim 3, for example, reciting 'hoisting ... candidate instructions within the nodes ...' does not establish a clear teaching how the hoisting is accomplished, in terms of the above source-destination paradigm. Likewise, claim 3 recites 'sinking ... candidate instructions within the nodes of the CFG loop' does not establish the above spatial requirement. One of ordinary skill in the art would not learn how hoisting can achieve *reduce* (re claim 1: 'reduce an amount of instructions between corresponding pairs of ... instructions ... modified CFG loop') of code from within a portion bounded by say, 'await' and/or 'advance' instructions, when *hoisting* does not teach instructions being actually removed away from that bounded loop portion; nor how *sinking* can achieve reducing of some block of CFG, absent any specific about how the sinking is helping reduce usefully one part of said CFG. Attempting to understand the claim in light of the Specifications, this above lack of teaching appears not so strongly remedied by the Disclosure. The Specifications does show insertion of AWAIT and ADVANCE instructions for bounding a

loop section but does not provide a clear scenario exploiting this bounding technique in such details about how *reducing* is achieved in terms of implementing removal of code with source-destination implication so characteristic of a hoist-and-sink concept. Conventionally, after a determination is made about which specific instruction to be eliminated, a process of elimination (such as in a hoisting or sinking) would have to clearly show the mechanics of **code removal conveying thereby a reduction in some form**. The claimed approach construed from the recited steps as to using a boundary instruction, queuing of block, processing candidate instruction do not amount to attaining code reduction; that is, for example, how the boundary instructions support code elimination? How is the queueing effectuated in light of the boundary instruction creation? What process is actually referencing --or storing information about-- location of final destination of evicted instructions. By not conveying how (e.g. spatial movement of code) as a program flow (as a loop) can be reduced via *hoisting* or *sinking* (i.e. *candidate instructions ... among* the basic blocks) in manner that these terms are conventionally understood, the claim amounts to a lack of essential teaching; thus, claims 3, 13 are rejected for not providing essential teachings enabling one of ordinary skill in the art to make use of the invention as it is claimed, without undue experimentation. Moreover, one's interpretation can be that the claimed invention cannot yield a useful result because code is not construed as being reduced. A lack of useful result issue would then be perceived as a **non-statutory** subject matter. The *sinking* and *hoisting* step will be treated with as broad a connotation as it can be reasonably possible – e.g. a general instruction reordering.

Claims 4-9, 14-19 and 33, 36 for reciting the same *hoisting* and *sinking* without any insight on how such can reduce the loop recited in the base claims are equally rejected for lack of

essential elements in the claimed subject matter, rendering the use or making of this invention a unduly hard undertaking for one of ordinary skill in the art.

Claim Rejections - 35 USC § 102

8. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

9. Claims 1, 11 are rejected under 35 U.S.C. 102(b) as being anticipated by Gupta et al, 6.044,221 (hereinafter Gupta).

As per claim 1, Gupta discloses method comprising:

building a control flow graph (CFG) for a loop body of a sequential application program to form a CFG loop (e.g. Fig. 5; acyclic graph, loop - col. 12, lines 45-54);

updating nodes of the CFG loop to enclose identified critical sections of the sequential application program within pairs of boundary instructions (e.g. PRES, DEAD, FREE, entry, exit col. 15, lines 10-65; *OnPath, UnAvailPaths, MayunAvail* - col. 17, line 57 to col. 18, line 67 – Note: instruction implemented at entry or exit node of a CFG reads on critical sections being identified for frequency of path usage; e.g. bit vectors associated with a unique path and being calculated at node exit reads on boundary instruction of frequently used by some unique regions); and

modifying nodes of the CFG loop to reduce an amount of instructions between corresponding pairs of bonding instructions (Fig. 8, 9, 11) to form a modified CFG loop.

As per claim 11, Gupta discloses an article of manufacture including a machine readable medium having stored thereon instructions which may be used to program a system to perform a method, comprising:

building a control flow graph (CFG) for a loop body of a sequential application program to form a CFG loop;

updating nodes of the CFG loop to enclose identified critical sections of the sequential application program within pairs of boundary instructions; and

modifying nodes of the CFG loop to reduce an amount of instructions between corresponding pairs of bonding instructions to form a modified CFG loop;

all of which limitations having been addressed in claim 1.

10. Claims 21-32, 34-35 are rejected under 35 U.S.C. 102(b) as being anticipated by Sohi et al, "Multiscalar Processors", 1995, ACM 0-89791-698; pp. 414-425 (hereinafter Sohi).

As per claim 21, Sohi discloses a method comprising: partitioning a sequential application program into a plurality of application program threads (Introduction, pg. 414); and concurrently executing the plurality of application program threads within a respective thread of a multi-threaded architecture (*...partitioned into tasks* -- sec 2.1, pg. 415, L col; Fig. 2; sec 2.2).

As per claim 22, Sohi discloses wherein partitioning the sequential application program comprises:

determining a thread count of a multi-threaded architecture; receiving identified critical sections within the sequential application program (e.g. L col. pg. 417 to R col. pg. 417 – Note: wait and forward instruction reads on critical regions being identified); and generating a plurality

of application program threads according to the thread count (sec 2.1; 2.2 pg. 415-417 – Note: partition of task in light of algorithm that each processing unit investigate variables resources – Fig. 3 – or investigate mask for register availability reads on taking into account thread count in a algorithmic approach to support thread synchronization) to synchronize access to identified critical sections among the plurality of application program threads.

As per claim 23, Sohi discloses concurrently executing further comprises: executing each iteration of a thread program loop by distinct multi-threaded architecture (Introduction, pg. 414; sec 3, pg. 419); and executing critical sections (re claim 22) of the thread program loop sequential thread order (e.g. *forward and release, existing binary* … - pg. 418, L col).

As per claim 24-25, Sohi discloses processing identified critical sections to reduce an amount of code (e.g. *non-useful … squashed … next unit … redirected, change structure of … loop …exit occurs* sec 3.1.2, pg. 420) contained within critical sections of thread program loops, such as code motion (*… next unit … redirected, change structure of … loop …exit occurs* sec 3.1.2, pg. 420).

As per claim 26, Sohi discloses article of manufacture including a machine readable medium having stored thereon instructions which may be used to program a system to perform a method, comprising: partitioning (a sequential application program); and concurrently executing (… respective thread of a multi-threaded architecture);

all of which having been addressed in claim 21.

As per claims 27-30, refer to claims 22-25.

As per claim 31, Sohi discloses an apparatus, comprising: a processor; a memory coupled to the processor, the memory including a compiler to cause a partition a sequential

application program into a plurality of application program-threads to enable concurrent execution of the plurality of program-threads within a respective thread of a multi-threaded architecture;

all of which having been addressed in claim 21.

As per claim 32, Sohi discloses wherein the compiler to cause building a control flow graph (CFG) for a loop body of a sequential application program to form a CFG (e.g. Fig. 2) loop to cause an update of nodes of the CFG loop to enclose identified critical sections (e.g. L col. pg. 417 to R col. pg. 417 – Note: *wait* and *forward* instruction reads on critical regions being identified) of the sequential application program within pairs of boundary instructions and to cause modification of nodes of the CFG loop to reduce an amount of instructions between corresponding pairs of bonding instructions (e.g. *non-useful ... squashed ... next unit ... redirected*, *change structure of ... loop ...exit occurs* sec 3.1.2, pg. 420) to form a modified CFG loop.

As per claims 34-35, refer to corresponding rejections set forth in claims 31-32, respectively.

Claim Rejections - 35 USC § 103

11. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

12. Claims 2-10, 12-20 are rejected under 35 U.S.C. 103(a) as being unpatentable over Gupta et al, USPN: 6,044,221, in view of APA (BACKGROUND of INVENTION: pg. 1, bottom), and further in view of Sohi et al, "Multiscalar Processors", 1995, ACM 0-89791-698; pp. 414-425.

Gupta discloses updating the CFG loop comprising: selecting an identified critical section of the sequential application program; inserting an instruction within a top node of the CFG loop; inserting an instruction within a bottom node of the CFG loop (re claim 1); and repeating the selecting, inserting and inserting for each identified critical section of the sequential application program (*OnPath, UnAvailPaths, MayunAvail* - col. 17, line 57 to col. 18, line 67; Fig. 8-11); but does not explicitly disclose that the top node instruction is a await instruction, nor is the bottom node instruction a advance instruction.

Gupta, however, disclose using heuristics with vectorization for determining frequency of execution related to hoist-sink optimization of a loop, such that this vector-based computation enable simultaneously calculate cost from entering a CFG unique path or exiting it (see col. 21, lines 24-32; Fig. 4-5, 8-11). Gupta's simultaneous dependency of a many interdependent instructions that require vector technique so to enable simultaneous execution of instructions while addressing dependency of data thereof was a known concept as expressed by **APA** via use of computer architecture with multi-threaded capability to transform sequential program to be run at parallelism level as suggested above. Dependency of concurrent instructions in a parallel-threaded architecture or environment is exemplified in **Sohi**'s VLIW multi-processing; according to which, multiscalar model of execution uses a CFG --analogous to Gupta-- to inspect instruction dependency per tasks per processor for re-ordering based on path inspection (see pg. 414, R column; col. 415, R col; re-ordered – sec 4.1) and one such task being inspected is

coordinated with bit or mask (similar to Gupta's bit vector) implemented as an *instruction* at the beginning of a CFG region just so said instruction would indicate a *wait* or a *forward* scenario dependent on whether resources are properly allocated for the task being inspected (see sec 2.2: pg. 417, L col to R col). In case where the vectorization technique by Gupta be used in a multi-scalar as taught by APA as established above, and based on the dependency of data or register value (see Gupta: Fig. 13) needed for a concurrent thread execution as implicated therein, it would have been obvious for one skill in the art at the time the invention was made to implement the boundary instruction by Gupta using the bit/mask instruction as by Sohi, so that one such boundary instruction would indicate a WAIT instruction or a ADVANCE instruction because this would help support the resolution of highly competitive critical region or data dependency therein, as mentioned by Gupta and further enhanced by Sohi, in light of the highly multi-threaded contention problem posed by exploiting multi-scalar execution as set forth by APA.

As per claim 3, Gupta (combined with APA and Sohi) discloses hoisting identified motion candidate instructions within the nodes of the CFG loop using code motions with fixed await boundary instructions; sinking identified motion candidate instructions within the nodes of the CFG loop using code motion with fixed advance instructions; and hoisting identified motion candidate instructions within the nodes of the CFG loop with fixed await instructions and fixed advance instructions (e.g. col. 5, line 23-50; col. 9-10; Fig 7-8).

As per claim 4, Gupta (combined with APA and Sohi) discloses

identifying every instruction within a basic block the CFG loop, excluding await instructions, as a motion candidate instructions (re claim 1 – Note: based on some computation about a expression about entry/exit of a path – Fig. 9 -- analyzing a CFG for node instruction to

be qualified for a sink or a hoist read on identifying every instruction excluding the boundary or predicate instruction);

building an inverse graph of the CFG loop (backward data flow analysis – col. 22, lines 15-35);

hoisting motion candidate instructions of the basic blocks until hoist instructions are no longer detected from the motion candidate instructions; and hoisting detected hoist instructions from motion candidate instructions in a source basic block of the CFG loop according to a dependence graph of the sequential application program (Note: heuristics to compute cost based on vector bits along paths -- see col. 17-18; Fig 7 – reads on detecting all candidate motion instruction for hoist according to the dependency being calculated via heuristics and inverse graph cost estimation technique - col. 22, lines 15-35).

Gupta discloses the backward cost estimation analysis having the basic blocks ordered according to a topological order indicated by the inverse graph; but does not explicitly teach 'initializing a hoist queue with the basic blocks from the CFG loop'; but based on the heuristics in traversing a graph with evaluating cost analysis (col. 22, lines 5-35; Fig. 11) visiting each basic block n of a CFG, the iteration to obtain summation in cost from the path being traversed suggests a set of node (a basic node in a acyclic tree) being queued for computing algorithm to be carried out. It would have been obvious for one skill in the art at the time the invention was made to implement the computation by Gupta so that the inverse tree cost analysis would include with a queue structure identifying which node (basic block) has already being visited for a path to be inspected as candidate for internal hoisting, as set forth above.

As per claim 5, Gupta discloses computing hoist instructions from the motion candidate instructions of the basic blocks based on a dependence graph of the sequential application program; hoisting the computed hoist instructions into a corresponding basic block (see Fig .7, re claim 1); but does not explicitly teach de-queuing a basic block from the hoist queue as a current block; and enqueuing the current block's predecessors from the CFG loop into the hoist queue when a change is detected. But in light of the above rationale (re claim 4) as to queuing all nodes being traversed in order to reorder a path portion bounded by the predicate instruction (re claim 1; Fig. 4-5), it would have been obvious for Gupta to queue CFG blocks prior to analyzing potential hoist possibility, and with the code being rearranged as a result of such hoisting, reordering the CFG including re-enqueing the reorder node of the tree to carry out the analysis of the modified tree, so to maintain instruction time-based sequencing or adjusted flow and resource dependency of such flow (e.g. Fig 13).

As per claims 6-7, Gupta (in view of the obviousness rationale of claim 4-5; see Fig. 7-8) discloses wherein sinking detected sink instructions with fixed advance instructions comprises: identifying motion candidate instructions within the basic blocks of the CFG loop through dataflow analysis with fixed advance instructions; initializing a sink queue with the basic blocks ordered based on a topological order in the CFG loop; sinking detected sink instructions among the basic blocks until sinking instructions are no longer detected; and sinking detected motion candidates within basic blocks that contain advance instructions according to the dependence graph;

de-queue a basic block from the sink queue as a current block; computing sink instructions from motion candidate instructions based on a dependence graph of the sequential

application program; sinking computed sink instructions into a corresponding basic block; and en-queuing a current block's successors in the CFG loop into the sink queue if a change is detected.

As per claim 8, Gupta (in view of the obviousness rationale of claim 4-7) discloses initializing a hoist queue with the basic blocks ordered based on a topological order in the CFG loop; identifying motion candidate instructions within the basic blocks of the CFG loop through dataflow analysis with fixed advance instructions and fixed await instructions; hoisting detected hoist instructions among the basic blocks until hoist instructions are no longer detected; and hoist motion candidates within basic blocks that contain await instructions based on a dependence graph of the sequential application program.

As per claim 9, Gupta (in view of the obviousness rationale of claim 4-7) discloses wherein motion candidate instructions hoisted out of an outmost await instruction are no longer treated as motion candidates; and wherein motion candidate instructions out of an outmost advance instruction are no longer treated as motion candidates (Note: Gupta approach of using heuristics in order to iteratively address a path based on computation being implemented in such path traversal – see Fig. 4-5; Fig. 8-11 – does read on exclusion of instruction from such computation once such instruction has been removed from the basic block under inspection).

As per claim 10, the concept as to addressing simultaneous data dependency in a multi-instruction execution as well known in the art such that highly multi-scalar architecture (see APA, Sohi) can be used in code synchronizing and reordering via CFG analysis has been visited in claims 2-3; hence the limitation as to:

forming a plurality of application program thread partitions from the modified CFG loop; and concurrently executing the plurality of application program threads within a respective thread of a multi-threaded architecture

would have been obvious for the same reasons as set forth in claims 2-3, above.

As per claim 12, refer to the rationale of claim 2.

As per claims 13-20, refer to the respective rejection as set forth in claims 3-10.

13. Claims 33, 35 are rejected under 35 U.S.C. 103(a) as being unpatentable over Sohi et al, "Multiscalar Processors", 1995 in view of Gupta et al, USPN: 6,044,221.

As per claims 33 and 36, Sohi discloses reordering of loops based on critical areas bounded by special instructions (e.g. L col. pg. 417 to R col. pg. 417 – i.e. fixed await and fixed advance instructions as boundary instructions) with moving of instructions out of some locations (e.g. sec 3.1.2, pg. 420); but does not explicitly teach:

hoisting identified motion candidate instructions within the nodes of the CFG loop using code motions with fixed await boundary instructions; and *sinking* of identified motion candidate instructions within the nodes of the CFG loop using code motion with fixed advance instructions and to cause hoisting of motion candidate instructions within the nodes of the CFG loop with fixed await instructions and fixed advance instructions.

The reordering of node in a CFG by Sohi's approach (sec 2.2 , pg. 417) for analyzing resources synchronization and reduction of non-useful instructions (sec 3.1→3.2.2, pg. 419, 420) is equivalent to identifying what instructions is susceptible to be reordered out of a critical section of the DFG as set forth above. As far as reordering instructions, the methodology is

further disclosed in Gupta's CFG-based reduction of non-useful instructions. Gupta discloses calculating a cost of a path when traversing such CFG for evaluation critical regions (e.g. PRES, DEAD, FREE, entry, exit col. 15, lines 10-65; *OnPath, UnAvailPaths, MayunAvail* - col. 17, line 57 to col. 18, line 67; Fig. 13-14), and further teaches *hoist* and *sink* (e.g. Fig. 7, Fig. 8) associated with the reordering and redirecting of code in and out of a given path. It would have been obvious for one skill in the art at the time the invention was made to implement *hoist* and *sink* as taught by Gupta based on bounded regions of the CFG as by Sohi where boundary instructions (wait, forward) enable analysis of useful or non-useful resources demand within specific parts of those critical and bounded regions, because according to the scheme of obviating non-useful resources and anticipating what resources demand precedences are in multiscalar operation by Sohi (see sec 3, pg. 419-420), Gupta's hoisting of code can support anticipation of what is needed first, and Gupta's sinking can support redirecting instructions that might not require execution within one critical area where resources are targeted for more urgent tasks as endeavored by Sohi.

Conclusion

14. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tuan A Vu whose telephone number is (571) 272-3735. The examiner can normally be reached on 8AM-4:30PM/Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Meng-Ai An can be reached on (571)272-3756.

The fax phone number for the organization where this application or proceeding is assigned is (571) 273-3735 (for non-official correspondence - please consult Examiner before

using) or 571-273-8300 (for official correspondence) or redirected to customer service at 571-272-3609.

Any inquiry of a general nature or relating to the status of this application should be directed to the TC 2100 Group receptionist: 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Tuan Anh Vu

Tuan A Vu
Patent Examiner,
Art Unit 2193
July 18, 2007